修改 HTML 元素

方法	描述
document.createElement(*element*)	创建 HTML 元素节点。
document.createAttribute(*attribute*)	创建 HTML 属性节点。
document.createTextNode(*text*)	创建 HTML 文本节点。
元素节点.removeChild(*element*)	删除 HTML 元素。
元素节点.appendChild(*element*)	添加 HTML 元素。
元素节点.replaceChild(*element*)	替换 HTML 元素。
元素节点.insertBefore(*element*)	在指定的子节点前面插入新的子节点。

1.节点-创建:

```
<div class="mydiv">9000</div>
<script>
    //创建了一个新的元素节点
    let newNode = document.createElement('h1');
    newNode.innerText = '大标题';
    console.dir(newNode);
    //把该节点追加到body里面-->append-->追加在最后面
    document.body.append(newNode);
    //把该节点追加到指定的节点里面-->append-->追加在最后面
    //一个节点只能在一个地方显示
    document.getElementsByClassName('mydiv')[0].append(newNode);
</script>
```

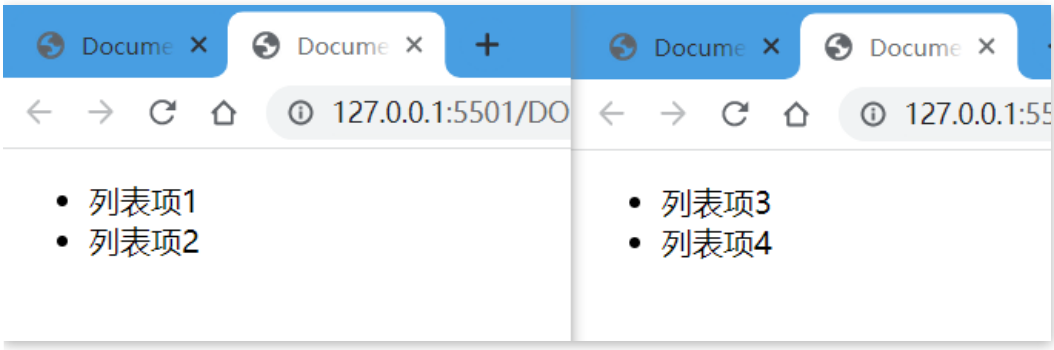
批量创建:

```
<div class="demo"></div>
    <script>
        let data = [
            "笔记本电脑主机-淘宝热卖排行,品质好货,快速到家!",
            "便携电脑 [京东] 高品质,电脑整机,运行体验更畅快!",
            "成都 小熊U租租电脑的平台的,企业一天一台起租",
            "电脑 - 商品 - 全网热卖",
            "太平洋电脑网_专业IT门户网站",
```

```
            "戴尔Premier会员购-dell笔记本-立即注册享好价!",
            "电脑(计算机) - 百度百科"
        ];

        // forEach循环创建
        let demo = document.querySelector('.demo');
        let str = '';
        data.forEach(item => {
            str += `<h2>${item}</h2>`;
        })
        demo.innerHTML = str;

        // 虚拟DOM
        let allH2 = document.createDocumentFragment();
        data.forEach(item => {
            let h2 = document.createElement('h2');
            h2.innerHTML = item;
            allH2.append(h2);
        })
        demo.append(allH2)
    </script>
```

```
//方法1
let ul = document.createElement('ul');
let li1 = document.createElement('li');
let text1 = document.createTextNode('列表项1');
li1.appendChild(text1);
ul.appendChild(li1);

//方法2
let li2 = document.createElement("li");
li2.innerHTML = "列表项2";
ul.appendChild(li2);

//方法3
let li3 = "<li>列表项3</li>";
let li4 = "<li>列表项4</li>";
ul.innerHTML = li3 + li4;
```

```
17
18 document.getElementsByTagName('body')[0].appendChild(ul);
```



2.节点-追加到指定的位置：

```
1  <ul>
2      <li>节点1</li>
3      <li>节点2</li>
4      <li>节点3</li>
5      <li id="fourth">节点4</li>
6      <li>节点5</li>
7  </ul>
8  <script>
9      let li = document.createElement('li');
10     li.innerText = '新的节点';
11
12     //插入到指定的节点前面 fourth：是getElementById('fourth')的简写
13     document.querySelector('ul').insertBefore(li, fourth);
14
15     //在指定的节点后面插入一个新的节点-->insertBefore-->node.nextElementSibling
16     let newli = document.createElement('li');
17     newli.innerText = '在4的后面插入一个新的节点';
18     document.querySelector('ul').insertBefore(newli, fourth.nextElementSibling);
19 </script>
```

- 节点1
- 节点2
- 节点3
- 新的节点
- 节点4
- 在4的后面插入一个新的节点
- 节点5

3.节点-复制：

```
<ul>
    <li>节点1</li>
    <li>节点2</li>
    <li>节点3</li>
    <li id="fourth">节点4</li>
    <li>节点5</li>
</ul>
 <hr>

 <script>
     let ul = document.querySelector('ul');
     //是否采用深度克隆,如果为true,则该节点的所有后代节点也都会被克隆,如果为false,则只克隆该节点本身.
     let newNode = ul.cloneNode(true);
     console.dir(newNode);
     document.body.appendChild(newNode);
 </script>
```

- 节点1
- 节点2
- 节点3
- 节点4
- 节点5

- 节点1
- 节点2
- 节点3
- 节点4
- 节点5

4.节点-删除

```
<ul>
    <li>节点1</li>
    <li>节点2</li>
    <li>节点3</li>
    <li id="fourth">节点4</li>
    <li>节点5</li>
</ul>
<hr>

<script>
    //通过父级删除指定的子节点
    let ul = document.querySelector('ul');
    let fourth = document.getElementById('fourth');
     ul.removeChild(fourth);

     //通过节点删除自己
     document.getElementsByTagName('li')[3].remove();
 </script>
```

- 节点1
- 节点2
- 节点3

5.节点-replace:

```
<ul>
    <li>节点1</li>
    <li>节点2</li>
    <li>节点3</li>
    <li id="fourth">节点4</li>
    <li>节点5</li>
</ul>

<script>
     //通过父级 替换 指定的子节点
     let ul = document.querySelector('ul');
     let fourth = document.getElementById('fourth');
     //需要创建一个新的节点
     let newli = document.createElement('li');
     newli.innerHTML = '这是替换后的节点';
     ul.replaceChild(newli, fourth);
     //新节点在前，被替换节点在后
</script>
```

动态判断、添加、删除、切换样式：

```
 <style>
        .b1 {
            width: 100px;
            height: 100px;
            text-align: center;
            line-height: 100px;
            background-color: pink;
        }

        .b2 {
            width: 300px;
            height: 300px;
            background-color: blue;
        }
    </style>

</head>
```

```
18
19  <body>
20      <button id="btn0">判断b2样式</button>
21      <button id="btn1">添加b2样式</button>
22      <button id="btn2">删除b2样式</button>
23      <button id="btn3">切换b2样式</button>
24
25      <br>
26      <br>
27
28      <div id="box" class="b1">b1</div>
29
30      <script>
31          let btn0 = document.getElementById('btn0');
32          let btn1 = document.getElementById('btn1');
33          let btn2 = document.getElementById('btn2');
34          let btn3 = document.getElementById('btn3');
35          let box = document.getElementById('box');
36
37          btn0.onclick = function () {
38              alert(hasClass(box, 'b2'));
39          };
40          btn1.onclick = function () {
41              addClass(box, 'b2');
42          }
43          btn2.onclick = function () {
44              removeClass(box, 'b2');
45          }
46          btn3.onclick = function () {
47              toggleClass(box, 'b2');
48          }
49
50          function hasClass(obj, cn) {
51              let reg = new RegExp('\\b' + cn + '\\b');
52              return reg.test(obj.className);
53          }
54
55          function addClass(obj, cn) {
56              if (!hasClass(obj, cn)) {
57                  obj.className += '' + cn;
```

```
            }
        }

        function removeClass(obj, cn) {
            let reg = new RegExp('\\b' + cn + '\\b');
            obj.className = obj.className.replace(reg, '');
        }

        function toggleClass(obj, cn) {
            if (hasClass(obj, cn)) {
                removeClass(obj, cn);
            } else {
                addClass(obj, cn);
            }
        }
    </script>
```

6.三种创建元素节点方式的区别

```
<body>
    <button>点击</button>
    <p>abc</p>
    <div class="inner"></div>
    <div class="create"></div>
```

```
    <script>
        // window.onload = function() {
        //         document.write('<div>123</div>');

        //     }
        // 三种创建元素方式区别

        // 1. document.write() 创建元素  如果页面文档流加载完毕，再调用这句话会导致页面重绘
        // var btn = document.querySelector('button');
        // btn.onclick = function() {
        //     document.write('<div>123</div>');
        // }

        // 2. innerHTML 创建元素
        var inner = document.querySelector('.inner');
        // for (var i = 0; i <= 100; i++) {
        //     inner.innerHTML += '<a href="#">百度</a>'
        // }
        var arr = [];
        for (var i = 0; i <= 100; i++) {
            arr.push('<a href="#">百度</a>');
        }
        inner.innerHTML = arr.join('');

        // 3. document.createElement() 创建元素
        var create = document.querySelector('.create');
        for (var i = 0; i <= 100; i++) {
            var a = document.createElement('a');
            create.appendChild(a);
        }
    </script>
```

7.效率测试

```
// 1 innerHTML拼接效率测试
function fn() {
        var d1 = +new Date();

```

```
        var str = '';
        for (var i = 0; i < 1000; i++) {
            document.body.innerHTML += '<div style="width:100px; height:2px; border:1px solid blue;"></div>';
        }
        var d2 = +new Date();
        console.log(d2 - d1);
    }
    fn();

// 2 innerHTML数组效率测试
 function fn() {
        var d1 = +new Date();
        var array = [];
        for (var i = 0; i < 1000; i++) {
            array.push('<div style="width:100px; height:2px; border:1px solid blue;"></div>');
        }
        document.body.innerHTML = array.join('');
        var d2 = +new Date();
        console.log(d2 - d1);
    }
    fn();

// 3 createElement效率测试
 function fn() {
        var d1 = +new Date();

        for (var i = 0; i < 1000; i++) {
            var div = document.createElement('div');
            div.style.width = '100px';
            div.style.height = '2px';
            div.style.border = '1px solid red';
            document.body.appendChild(div);
        }
        var d2 = +new Date();
        console.log(d2 - d1);
    }
    fn();

```

```
区别：
1 document.write是直接将内容写入页面的内容流，但是文档流执行完毕，则它会导致页面全部重绘，innerHTML是将内容写入某个 DoM节点，不会导致页面全部重绘
2 innerHTML创建多个元素效率更高(不要拼接字符串，采取数组形式拼接），结构稍微复杂
3 createElement( )创建多个元素效率稍低一点点，但是结构更清晰
 总结：不同浏览器下，innerHTML效率要比 creatElement高
```

事件:

0. 事件绑定

（1）行内绑定方式：在标签行内的时间值上写上标志"javaScript:后跟js代码"

```
<a href="javaScript:alert(666)">点我</a>
<a href="javaScript:void(0)">点我</a>
<button onclick="javaScript:alert(666)">点我</button>
```

（2）元素属性绑定方式

```
el.onxxx=function(event){}
// 兼容性很好，但是一个元素的同一个时间上只能绑定一个处理程序
// 句柄式写法，基本等同于写在HTML行间上

obj.onclick = method1;
obj.onclick = method2;
obj.onclick = method3;
// 以上代码只有method3执行
```

（3）同元素多处理程序绑定方式1 (可通过这种方式绑定多次)

```
obj.addEventListener(type,fn,false);
// IE9以下不兼容，可以为一个事件绑定多个处理程序
// 第三个参数表示是否捕获阶段触发,跟冒泡没关系(笔试陷阱题)

btn1Obj.addEventListener('click',method1,false);
btn1Obj.addEventListener('click',method2,false);
btn1Obj.addEventListener('click',method3,false);
// 以上代码执行顺序是method1、method2、method3
```

（4）同元素多处理程序绑定方式2(了解)

```
obj.attachEvent('on'+type,fn);
// IE专有( ie11例外)，一个事件同样可以绑定多个处理程序
// <meta http-equiv="X-UA-Compatible" content="IE=9" />，解决 IE11向后兼容 IE9的问题

btn1Obj.attachEvent("onclick",method1);
btn1Obj.attachEvent("onclick",method2);
btn1Obj.attachEvent("onclick",method3);
// 注意的是需要加on，比如onclick,onsubmit,onchange，执行顺序是method3、method2、method1
// 这是微软的私人方法，火狐和其他浏览器都不支持


// 兼容解决
<script>
    function addEvent(el,ev,fn,useCapture){
        if (el.addEventListener) {
            el.addEventListener(ev,fn,useCapture);//DOM 2.0
            return true;
        }else if(el.attachEvent){
            var r = el.attachEvent('on'+ev,fn); //IE5+
            return r;
        }else{
            el['on'+ev] = fn; //DOM 0
        }
    }
</script>
```

（5）多元素同事件同处理程序绑定方式==>委托（代理）模式

```
box.onclick = function (e) {
    e.target
}
// 父元素绑定事件  通过事件对象来区分用户触发的事件属于哪一个具体的对象
```

1.事件注册的两种方法

```
<body>
    <button>传统注册事件</button>
```

```
    <button>方法监听注册事件</button>
    <button>ie9 attachEvent</button>
    <script>
        var btns = document.querySelectorAll('button');
        // 1. 传统方式注册事件
        btns[0].onclick = function() {
            alert('hi');
        }
        btns[0].onclick = function() {
                alert('hao a u');
            }
        // 2. 事件侦听注册事件 addEventListener
        // (1) 里面的事件类型是字符串 必定加引号 而且不带on
        // (2) 同一个元素 同一个事件可以添加多个侦听器（事件处理程序）
        btns[1].addEventListener('click', function() {
            alert(22);
        })
        btns[1].addEventListener('click', function() {
                alert(33);
            })
            // 3. attachEvent ie9以前的版本支持
        btns[2].attachEvent('onclick', function() {
            alert(11);
        })
    </script>
</body>
```

2.删除事件（事件解绑）

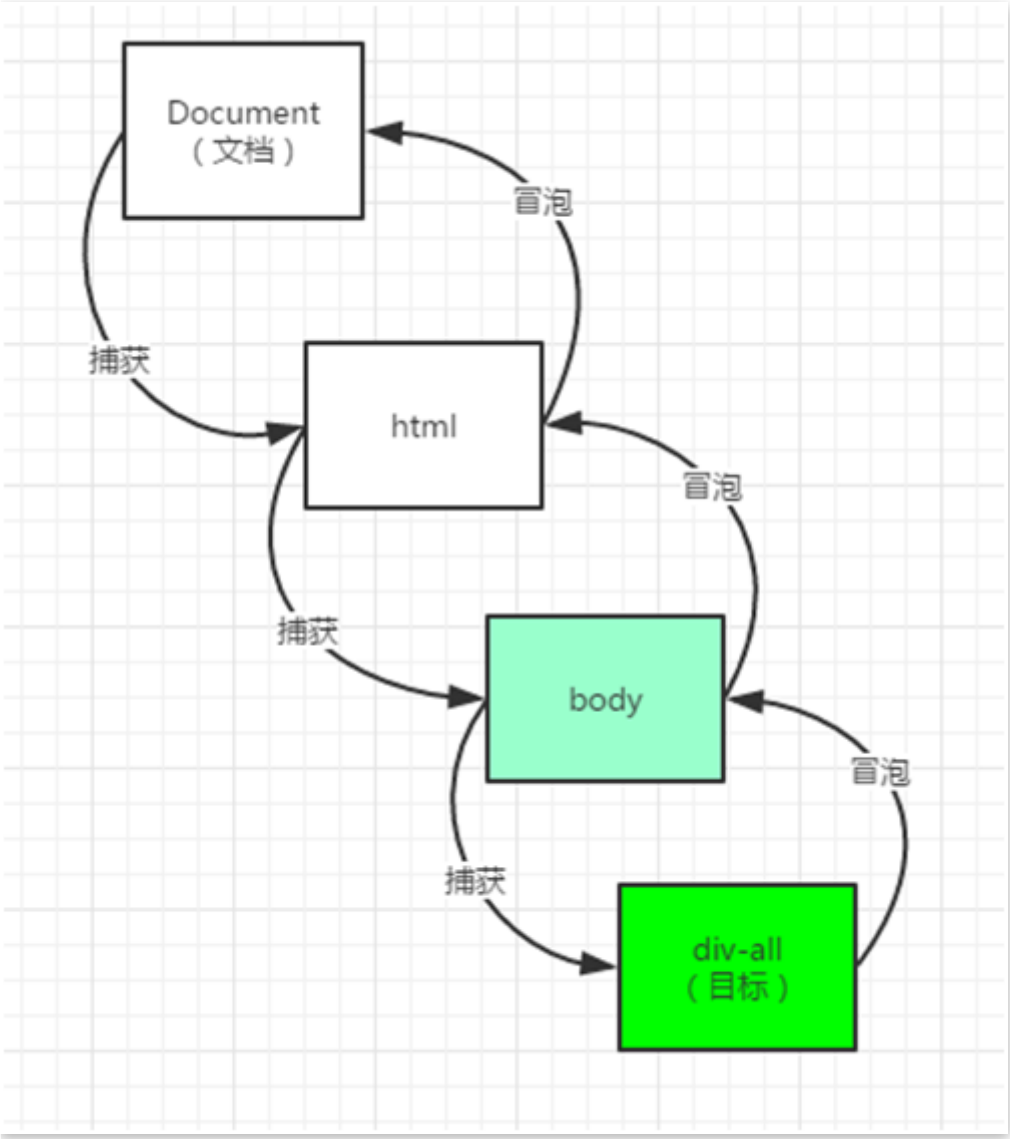
- el.onclick= false / '' / null;
- el.removeEventListener(type, fn, false);
- el.detachEvent("on"+type, fn);

提示: 2,3若绑定的是匿名函数，则永远无法解除

```
<body>
    <div>1</div>
    <div>2</div>
    <div>3</div>
    <script>
```

```
        var divs = document.querySelectorAll('div');
        divs[0].onclick = function() {
                alert(11);
                // 1. 传统方式删除事件
                divs[0].onclick = null;
            }
            // 2. removeEventListener 删除事件
        divs[1].addEventListener('click', fn) // 里面的fn 不需要调用加小括号

        function fn() {
            alert(22);
            divs[1].removeEventListener('click', fn);
        }
        // 3. detachEvent
        divs[2].attachEvent('onclick', fn1);

        function fn1() {
            alert(33);
            divs[2].detachEvent('onclick', fn1);
        }
    </script>
</body>
```

3.DOM事件流三个阶段



1.事件的三个阶段:

先捕获，后目标，再冒泡,只能有一个阶段触发程序执行,比如捕获阶段触发了到了冒泡阶段就不再触发事件经过所有元素都没有被处理,这个事件消失

事件传递的过程 只跟文档树的结构有关系 跟界面显示的层叠效果没有任何关系

- 事件冒泡：结构上（非视觉上）嵌套关系的元素，会存在事件冒泡的功能，即同一事件，自子元素冒泡向父元素。（自底向上）
- 事件捕获: 结构上（非视觉上）嵌套关系的元素，会存在事件捕获的功能，即同一事件，自父元素捕获至子元素（事件源元素）。（自顶向下）

2.默认在冒泡的时候执行事件: onclick/attach和 addEventListener默认情况下。

3.addEventListener绑定事件，如果把第三个参数设置为true，则在捕捉的时候执行事件(只有这种方式可以让事件在捕获阶段执行，句柄式式在冒泡阶段执行)

4.整个事件处理过程，会有个event事件对象在整个事件过程传播（W3C标准，ie8及其以下没有）

5.ie8以下不支持addEventListener，attach没有第三个参数

6.focus,blur,change,submit,reset,select等事件不冒泡

```
<body>
```

```
    <div class="father">
        <div class="son">son盒子</div>
    </div>
    <script>
        // dom 事件流 三个阶段
        // 1. JS 代码中只能执行捕获或者冒泡其中的一个阶段。

        // 2. onclick 和 attachEvent（ie） 只能得到冒泡阶段。

        /*
        3. 捕获阶段 如果addEventListener 第三个参数是 true
        那么则处于捕获阶段  document -> html -> body -> father -> son
        */
        // var son = document.querySelector('.son');
        // son.addEventListener('click', function() {
        //     alert('son');
        // }, true);

        // var father = document.querySelector('.father');
        // father.addEventListener('click', function() {
        //     alert('father');
        // }, true);

        // 4. 冒泡阶段 如果addEventListener 第三个参数是 false 或者
        //省略 那么则处于冒泡阶段  son->father->body->html->document
        var son = document.querySelector('.son');
        son.addEventListener('click', function () {
            alert('son');
        }, false);

        var father = document.querySelector('.father');
        father.addEventListener('click', function () {
            alert('father');
        }, false);

        document.addEventListener('click', function () {
            alert('document');
        })
    </script>
</body>
```

4.事件对象

```
<body>
    <div>123</div>
    <script>
        // 事件对象
        var div = document.querySelector('div');
        div.onclick = function(e) {
                // console.log(e);
                // console.log(window.event);
                // e = e || window.event;
                console.log(e);
            }
            // div.addEventListener('click', function(e) {
            //         console.log(e);
            //     })

        // 1. event 就是一个事件对象 写到我们侦听函数的 小括号里面 当形参来看
        // 2. 事件对象只有有了事件才会存在，它是系统给我们自动创建的，不需要我们传递参数
        // 3. 事件对象 是 我们事件的一系列相关数据的集合 跟事件相关的 比如鼠标点击里面就包含了鼠标的相关信息，鼠标坐标啊，如果是键盘事件里面就包含的键盘事件的信息 比如 判断用户按下了那个键
        // 4. 这个事件对象我们可以自己命名 比如 event 、 evt、 e
        // 5. 事件对象也有兼容性问题 ie678 通过 window.event 兼容性的写法
        //兼容写法： let event = ev || window.event;
    </script>
</body>
```

5.事件对象e.target

```
<body>
    <div>123</div>
    <ul>
        <li>abc</li>
        <li>abc</li>
        <li>abc</li>
    </ul>
    <script>
```

```
        // 常见事件对象的属性和方法
        // 1. e.target 返回的是触发事件的对象（元素）
        //    this 返回的是绑定事件的对象（元素）
        // 区别 :
        //  e.target: 点击了那个元素，就返回那个元素，
        //  this: 那个元素绑定了这个点击事件，那么就返回谁
        var div = document.querySelector('div');
        div.addEventListener('click', function(e) {
            console.log(e.target);
            console.log(this);

        })
        var ul = document.querySelector('ul');
        ul.addEventListener('click', function(e) {
                // 我们给ul 绑定了事件  那么this 就指向ul
                console.log(this);
                console.log(e.currentTarget);

                // e.target 指向我们点击的那个对象 谁触发了这个事件 我们点击的是li
    e.target 指向的就是li
                console.log(e.target);

            })
            // 了解兼容性
            // div.onclick = function(e) {
            //     e = e || window.event;
            //     var target = e.target || e.srcElement;
            //     console.log(target);

        // }
        // 2. 了解 跟 this 有个非常相似的属性 currentTarget  ie678不认识
    </script>
</body>
```

6、事件对象：鼠标事件触发

```
altKey 鼠标事件发生时，是否按下alt键，返回一个布尔
ctrlKey 鼠标事件发生时，是否按下ctrl键，返回一个布尔
metaKey  鼠标事件发生时，是否按下windows/commond键，返回一个布尔
```

```
shiftKey 鼠标事件发生时，是否按下shift键，返回一个布尔

pageX  鼠标点击位置相对于网页左上角的水平偏移量，也就是clientX加上水平滚动条的距离
pageY  鼠标点击位置相对于网页左上角的垂直偏移量，也就是clientY加上垂直滚动条的距离
clientX clientY 返回鼠标位置相对于 浏览器窗口左上角 的坐标，单位为像素(不包括body隐藏的 不会计算水平滚动条的距离)
screenX screenY 返回鼠标位置相对于 屏幕(电脑)左上角 的坐标，单位为像素
movementX movementY返回一个位移值，单位为像素，表示当前位置与上一个mousemove事件之间的距离
offsetX offsetY 相对于元素自己的x/y 跟它是否是定位的元素无关(鼠标点击位置相对于触发事件对象的水平距离)
x  y 与 clientX clientY 一样
```

7.事件对象：键盘事件触发

1. keydown：在键盘上按下某个键时触发。如果按住某个键，会不断触发该事件，但是 Opera 浏览器不支持这种连续操作。
2. keypress：按下某个键盘键并释放时触发。如果按住某个键，会不断触发该事件。
3. keyup：释放某个键盘键时触发。该事件仅在松开键盘时触发一次，不是一个持续的响应状态。
4. 三个事件执行顺序

keydown-->keypress-->keyup

5. keydown和 keypress的区别

- keydown可以获取特殊键盘的事件(除Fn)，对于字母按键，大小写情况下按键对应值都一样。
- keypress只可以响应字符类键盘按键（event.charCode）
- keydown常用于绑定操作类事件处理，keypress常用于绑定字符类事件处理

6. 事件对象的 keyCode altkey shiftkey ctrlkey

```
charCode/keyCode  键码值  key  键码
37左
38上
39右
40下
13enter

document.onkeydown=function(event){
    let key_code =  event.keyCode;

     // 判断是否按钮了shift键
     if(e.shiftKey){
        console.log("你按下了shift键");
```

```
        }

        // 是否按钮了ctrl键
        if(e.ctrlKey){
            console.log("你按下了ctrl键");
        }

        // 是否按钮了alt键
        if(e.altKey){
            console.log("你按下了alt键");
         }
}
```

8.事件对象阻止默认行为

```
<body>
    <div>123</div>
    <a href="http://www.baidu.com">百度</a>
    <form action="http://www.baidu.com">
        <input type="submit" value="提交" name="sub">
    </form>
    <script>
        // 常见事件对象的属性和方法
        // 1. 返回事件类型
        var div = document.querySelector('div');
        div.addEventListener('click', fn);
        div.addEventListener('mouseover', fn);
        div.addEventListener('mouseout', fn);

        function fn(e) {
            console.log(e.type);

        }
        // 2. 阻止默认行为（事件） 让链接不跳转 或者让提交按钮不提交
        // 默认事件—表单提交，a标签跳转，右键菜单等等
        var a = document.querySelector('a');
        a.addEventListener('click', function (e) {
            e.preventDefault(); //  dom 标准写法
        })
```

```
        // 3. 传统的注册方式
        a.onclick = function (e) {
            // 普通浏览器 e.preventDefault();  方法
            e.preventDefault();
            // 低版本浏览器 ie678  returnValue  属性
            // e.returnValue;
            // 我们可以利用return false 也能阻止默认行为 没有兼容性问题 特点:  return 后面的代码不执行了,  而且只限于传统的注册方式
            return false;
            alert(11);
        }
    </script>
</body>
```

9.阻止事件冒泡

```
 <style>
        .father {
            overflow: hidden;
            width: 300px;
            height: 300px;
            margin: 100px auto;
            background-color: pink;
            text-align: center;
        }

        .son {
            width: 200px;
            height: 200px;
            margin: 50px;
            background-color: purple;
            line-height: 200px;
            color: #fff;
        }
    </style>
</head>

<body>
    <div class="father">
```

```
        <div class="son">son儿子</div>
    </div>
    <script>
        // 常见事件对象的属性和方法
        // 阻止冒泡  DOM 推荐的标准 stopPropagation()
        var son = document.querySelector('.son');
        son.addEventListener('click', function (e) {
            alert('son');
            e.stopPropagation(); // stop 停止  Propagation 传播
            e.cancelBubble = true; // 非标准 cancel 取消 bubble 泡泡
        }, false);

        var father = document.querySelector('.father');
        father.addEventListener('click', function () {
            alert('father');
        }, false);

        document.addEventListener('click', function () {
            alert('document');
        })
    </script>
</body>
```